Chapter 25
CoS Configuration Guidelines

To configure CoS properties, you can include the following statements at the [edit class-of-service] hierarchy level of the configuration:

```
class-of-service {
    classifiers {
        type classifier-name {
            import (classifier-name | default);
            forwarding-class class-name {
                loss-priority (low | high) code-points [ alias | bits ];
            }
        }
    }
    code-point-aliases {
        (dscp | exp | ieee-802.1 | inet-precedence) {
            alias-name bits;
        }
    }
    drop-profiles {
        profile-name {
            fill-level percentage drop-probability percentage;
            interpolate {
                drop-probability value;
                fill-level value;
            }
        }
    }
    forwarding-classes {
        queue queue-number class-name priority (low | high);
    }
    forwarding-policy {
        next-hop-map map-name {
            forwarding-class class-name {
                next-hop [ next-hop-name ];
                lsp-next-hop [ lsp-regular-expression ];
            }
        }
        class class-name {
            classification-override {
                forwarding-class class-name;
            }
        }
    }
```

```
        interfaces
            interface-name {
                scheduler-map map-name;
                unit logical-unit-number {
                    classifiers {
                        (dscp | exp | ieee-802.1 | inet-precedence) (classifier-name | default);
                    }
                    forwarding-class class-name;
                    rewrite-rules {
                        (dscp | exp | inet-precedence) (rewrite-name | default);
                    }
                }
            }
        }
        rewrite-rules {
            (dscp | exp | inet-precedence) rewrite-name {
                import (rewrite-name | default);
                forwarding-class class-name {
                    loss-priority (low | high) code-point (alias | bits);
                }
            }
        }
        scheduler-maps {
            map-name {
                forwarding-class class-name scheduler scheduler-name;
            }
        }
        schedulers {
            scheduler-name {
                buffer-size (percent percentage | remainder);
                drop-profile-map loss-priority (low | high) protocol (non-tcp | tcp | any)
                         drop-profile profile-name;
                priority (low | high | strict-high);
                transmit-rate (rate | percent percentage | remainder | exact);
            }
        }
    }
```

The following RFCs define the standards supported by certain aspects of the CoS software:

RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*

RFC 2598, *An Expedited Forwarding PHB* (see also draft-ietf-diffserv-rfc2598bis-01.txt)

RFC 2597, *Assured Forwarding PHB Group*

The JUNOS software supports only two loss priorities and, by default, supports only one assured forwarding (AF) class, although you can configure more at the expense of other class types.

The following RFC is not supported:

RFC 2983, *Diffserv and Tunnels*

Hardware Capabilities and Limitations

Juniper Networks T-series platforms and M-series platforms with enhanced FPCs can use an expanded range of CoS capabilities as compared to M-series platforms that employ the earlier FPC model. Table 23 lists these differences between the original FPCs and the enhanced FPCs.

Table 23: CoS Hardware Capabilities and Limitations

Feature	M-series FPCs	M-series Enhanced FPCs	T-series FPCs	Comments
Classifiers				
Limit per FPC	1	8	64	For M-series enhanced FPCs, four classifiers are shared between DSCP and IP and four are shared between MPLS EXP and IEEE 802.1p.
dscp	no	yes	yes	For T-series FPCs, the loss priority cannot be arbitrarily decoded from the code point.
ieee-802.1p	no	yes	yes	For T-series FPCs, the loss priority cannot be arbitrarily decoded from the code point.
inet-precedence	yes	yes	yes	For M-series original FPCs and T-series FPCs, the loss priority cannot be arbitrarily decoded from the code point.
mpls-exp	yes	yes	yes	For T-series FPCs, the loss priority cannot be arbitrarily decoded from the code point. For M-series original FPCs, fixed classification bit 0 indicates loss priority; bits 1 and 2 indicate queue number (forwarding class).
Rewrite Markers				
Limit per FPC	none	none	64	
dscp	no	yes	yes	For T-series FPCs, you cannot use loss priority to select the rewrite code point. It is based on forwarding class only.
ieee-802.1	no	yes	yes	For M-series enhanced FPCs and T-series FPCs, fixed rewrite loss priority determines the value for bit 0; queue number (forwarding class) determines bits 1 and 2.
inet-precedence	yes	yes	yes	For T-series FPCs, you cannot use loss priority to select the rewrite code point. It is based on forwarding class only.
mpls-exp	yes	yes	yes	For T-series FPCs, you cannot use loss priority to select the rewrite code point. It is based on forwarding class only. For M-series original FPCs, fixed rewrite loss priority determines the value for bit 0; queue number (forwarding class) determines bits 1 and 2.
Queuing				
Priority	no	yes	yes	
Drop Profiles				
Limit per FPC	2	16	32	
Per queue	no	yes	yes	
Per loss priority	yes	yes	yes	
Per TCP bit	no	yes	yes	

This chapter includes the following sections:

Define Code-Point Aliases

A code-point alias is a name you assign to a set of DiffServ code-point (DSCP) bits. When you configure classes and define classifiers, you can refer to the code points by these alias names. You can configure user-defined classifiers in terms of alias names. If the value of an alias changes, it alters the behavior of any classifier that references that alias.

Table 24 shows the default mappings between the bit values and standard aliases. For example, it is widely accepted that the alias for DSCP 101110 is ef (expedited forwarding).

Table 24: Default DSCP Mappings

DiffServ Code Designator	Mapping
DSCP Code Points:	
ef	101110
af11	001010
af12	001100
af13	001110
af21	010010
af22	010100
af23	010110
af31	011010
af32	011100
af33	011110
af41	100010
af42	100100
af43	100110
be	000000
cs1	001000
cs2	010000
cs3	011000

DiffServ Code Designator	Mapping
cs4	100000
cs5	101000
nc1/cs6	110000
nc2/cs7	111000
MPLS EXP Code Points:	
be	000
be1	001
ef	010
ef1	011
af11	100
af12	101
nc1/cs6	110
nc2/cs7	111
IEEE 802.1 Code Points:	
be	000
be1	001
ef	010
ef1	011
af11	100
af12	101
nc1/cs6	110
nc2/cs7	111
Legacy IP Precedence Code Points:	
be	000
be1	001
ef	010
ef1	011
af11	100
af12	101
nc1/cs6	110
nc2/cs7	111

You use code-point aliases to do the following:

Define an alias for bits that currently have no alias

Define multiple aliases for the same bits

Redefine an alias name to mean a different set of bits than the default

To define a code-point alias, include the code-point-aliases statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
code-point-aliases {
    (dscp | exp | ieee-802.1 | inet-precedence) {
        alias-name bits;
    }
}
```

For example, you might set up the following configuration:

```
[edit class-of-service]
code-point-aliases {
    dscp {
        my1 110001;
        my2 101110;
        be 000001;
        cs7 110000;
    }
}
```

The sample configuration produces this mapping:

```
user@host#show class-of-service map name type dscp
 Map: name Type: dscp
    101110      ef/my2
    001010      af11
    001100      af12
    001110      af13
    010010      af21
    010100      af22
    010110      af23
    011010      af31
    011100      af32
    011110      af33
    100010      af41
    100100      af42
    100110      af43
    000001      be
    001000      cs1
    010000      cs2
    011000      cs3
    100000      cs4
    101000      cs5
    110000      nc1/cs6/cs7
    111000      nc2
    110001      my1
```

The following notes explain certain results in the mapping:

my1 110001:

110001 was not mapped to anything before, and my1 is a new alias.

Nothing in the default mapping table is changed by this statement.

my2 101110:

101110 is now mapped to my2 as well as ef.

be 000001:

be is now mapped to 000001.

The old value of be, 000000, is not associated with any alias. Packets with this DSCP value are now classified to the default forwarding class.

cs7 110000:

cs7 is now mapped to 110000, as well as nc1 and cs6.

The old value of cs7, 111000, is still mapped to nc2.

Configure Forwarding Classes

Forwarding classes replace output queues from the previous CoS configuration command set. You assign each forwarding class to an internal queue number by including the forwarding-classes statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
forwarding-classes {
    queue queue-number class-name priority (high | low);
    }
}
```

Note

You cannot commit a configuration that assigns the same forwarding class to two different queues.

Table 25 shows the four forwarding classes defined by default:

Table 25: Default Forwarding Classes

Queue	Forwarding Class Name
queue 0	best-effort
queue 1	expedited-forwarding
queue 2	assured-forwarding
queue 3	network-control

The following rules govern queue assignment:

If classifiers fail to classify a packet, the packet always receives the default classification to the class associated with queue 0.

The number of queues is dependent on the hardware plugged into the chassis. CoS configurations are inherently contingent on the number of queues on the system. Only two classes, best-effort and network-control, are actually referenced in the default configuration. The default configuration works on any platform.

CoS configurations that specify more queues than the platform can support are not accepted. The commit fails with a detailed message that states the total number of queues available.

All default CoS configuration is based on queue number. The name of the forwarding class that shows up when the default configuration is displayed is the forwarding class currently associated with that queue.

This is the default configuration for forwarding-classes:

```
[edit class-of-service]
forwarding-classes {
    queue 0 best-effort;
    queue 1 expedited-forwarding;
    queue 2 assured-forwarding;
    queue 3 network-control;
}
```

If you reassign the forwarding-class names, the best-effort forwarding-class name appears in the locations in the configuration previously occupied by network-control as follows:

```
forwarding-classes {
    queue 0 network-control;
    queue 1 assured-forwarding;
    queue 2 expedited-forwarding;
    queue 3 best-effort;
}
```

All the default rules of classification and scheduling that applied to queue 3 still apply. Queue 3 is simply now renamed best-effort.

In the current default configuration:

Only IP precedence classifiers are associated with interfaces.

The only classes designated are best-effort and network-control.

Schedulers are not defined for the expedited-forwarding or assured-forwarding classes.

You must make a conscious effort to classify packets to the expedited-forwarding or assured-forwarding class and define schedulers for these classes.

Override Fabric Priority Queuing

For T-series platforms only, you can override automatic fabric priority queuing. For egress interfaces, fabric priority queuing matches the queue priority you assign at the [edit class-of-service schedulers *scheduler-name*] hierarchy level. High-priority egress traffic is automatically assigned to high-priority fabric queues. Likewise, low-priority egress traffic is automatically assigned to low-priority fabric queues.

You can override the default fabric priority queuing of egress traffic by including the priority statement at the [edit class-of-service forwarding-classes queue *queue-number class-name*] hierarchy level:

```
[edit class-of-service forwarding-classes queue queue-number class-name]
priority (low | high);
```

Classify Packets by Behavior Aggregate

The simplest way to classify a packet is to use behavior aggregate classification. The DSCP or IP precedence bits of the IP header convey the behavior aggregate class information. The information might also be found in the MPLS EXP bits or IEEE 802.1p CoS bits.

Table 26 shows the default system classification scheme for the well-known DSCPs:

Table 26: Default Behavior Aggregate Classification

DSCP	Forwarding Class	PLP
ef	expedited-forwarding	low
af11	assured-forwarding	low
af12	“	high
af13	“	high
af21	best-effort	low
af22	“	low
af23	“	low
af31	“	low
af32	“	low
af33	“	low
af41	“	low
af42	“	low
af43	“	low
be	“	low
cs1	“	low
cs2	“	low
cs3	“	low
cs4	“	low
cs5	“	low
nc1/cs6	network-control	low
nc2/cs7	“	low
other	best-effort	low

All af classes other than af1*X* are mapped to best-effort, since RFC 2597 prohibits a node from aggregating classes. In effect, mapping to best-effort implies that the node does not support that class.

To define new classifiers for all code-point types, include the classifiers statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
classifiers {
    (dscp | exp | ieee-802.1 | inet-precedence) classifier-name {
        import [classifier-name | default];
        forwarding-class class-name {
            loss-priority (low | high) code-points [alias | bits];
        }
    }
}
```

A classifier takes a specified bit pattern as either the literal pattern or as a defined alias and attempts to match it to the type of packet arriving on the interface. If the information in the packet's header matches the specified pattern, the packet is sent to the appropriate queue, defined by the forwarding class associated with the classifier.

You can use any table, including the default, in the definition of a new classifier by including the import statement. The imported classifier is used as a template and is not modified. Whenever you commit a configuration that assigns a new *class-name* and loss-priority value to a code-point alias or set of bits, it replaces that entry in the imported classifier template. As a result, you must explicitly specify every code point in every designation that requires modification.

Note

If an interface is mounted on an original FPC, you can apply to the interface the default exp classifier only. If an interface is mounted on an enhanced FPC, you can define and apply to it a new exp classifier.

You can assign the classification map to a logical interface by including the forwarding-class statement in the following configuration:

```
[edit class-of-service]
interfaces {
    interface-name {
        unit logical-unit-number {
            forwarding-class class-name;
        }
    }
}
```

You can use interface wildcards for *interface-name* and *logical-unit-number.*

The dscp classifier classifies all incoming IPv4 packets, while the exp classifier handles MPLS packet classification.

Note

You cannot mix L2 and L3 classification on an interface. For the purposes of this configuration, MPLS is considered L3 classification.

Configure Scheduling Policy Maps

You use scheduling policy maps to configure the forwarding classes that represent packet queues and associate them with physical interfaces.

A *scheduler* configuration block specifies the buffer size, bandwidth, and priority for a queue. It also specifies the RED drop profile for packets that fall within specification and out of specification. To configure schedulers, include the schedulers statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
schedulers {
    scheduler-name {
        buffer-size (percent percentage | remainder);
        drop-profile-map loss-priority (low | high) protocol (non-tcp | tcp | any)
                drop-profile profile-name;
        priority (low | high | strict-high);
        transmit-rate (rate | percent percentage | remainder | exact);
    }
}
```

Once you define a scheduler, you can include it in a *scheduler map* that is used to map a specified forwarding class to a scheduler configuration:

```
[edit class-of-service]
scheduler-maps {
    map-name {
        forwarding-class class-name scheduler scheduler-name;
    }
}
```

When you have defined the *map-name*, you can associate it with an output interface:

```
[edit class-of-service]
interfaces {
    interface-name {
        scheduler-map map-name;
    }
}
```

Interface wildcards are supported.

You must configure each forwarding class in turn. The following default set of schedulers and scheduler maps is provided with the installation:

```
[edit class-of-service]
schedulers {
    network-control {
        transmit-rate percent 5;
        buffer-size percent 5;
        priority low;
        drop-profile-map loss-priority low protocol any drop-profile passive;
        drop-profile-map loss-priority high protocol any drop-profile aggressive;
    }
    best-effort {
        transmit-rate percent 95;
        buffer-size 95;
        priority low;
        drop-profile-map loss-priority low protocol any drop-profile passive;
        drop-profile-map loss-priority high protocol any drop-profile aggressive;
    }
}
scheduler-map default {
    forwarding-class best-effort scheduler best-effort;
    forwarding-class network-control scheduler network-control;
}
```

Configure RED Drop Profiles

RED drop profiles are associated with the forwarding classes and loss priorities from the scheduler-map you configured on the interface. To configure the drop profiles themselves, include the drop-profiles statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
drop-profiles {
    profile-name {
        fill-level percentage drop-probability percentage;
        interpolate {
            fill-level value;
            drop-probability value;
        }
    }
}
```

In this configuration, you include either the interpolate statement and its options, or the fill-level and drop-probability *percentage* values. These two alternatives enable you to configure either each individual drop probability at up to 64 fill-level/drop-probability paired values, or a profile represented as a series of line segments.

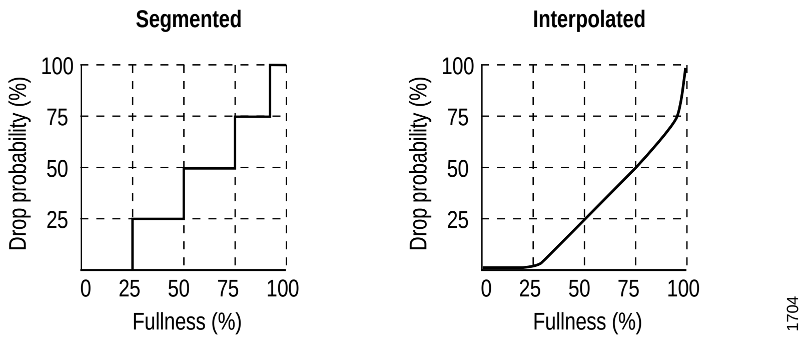
Note

If you configure the interpolate statement, you can specify more than 64 pairs, but the system generates only 64 discrete entries.

The line segments are defined in terms of the following graphical model: in the first quadrant, the x axis represents the fill level and the y axis represents the drop probability. The initial line segment spans from the origin (0,0) to the point (<l1>, <p1>); a second line runs from (<l1>, <p1>) to (<l2>, <p2>) and so forth, until a final line segment connects (100, 100). The system automatically constructs a drop profile containing 64 fill levels at drop probabilities that approximate the calculated line segments.

Figure 23 shows sample line graphs contrasting use of the segment percentages (on the left) and interpolated values (on the right):

Figure 23: Segmented and Interpolated Drop Profiles



Packet Loss Priority

The system supports two packet loss priority (PLP) designations, low and high.

The packet loss priority is used to determine the RED drop profile when queuing a packet. You can set it by configuring a classifier or policer.

Rewrite Packet Header Information

You can rewrite the packet header bits because the logical interface transmits the packet along with the forwarding-class and PLP information associated with the packet. The rewrite-rules configurations define the mappings.

Table 27 shows the default mappings.

Table 27: Default Packet Header Rewrite Mappings

Map To DSCP / EXP / IEEE / IP	Map From Forwarding Class	PLP Value
ef	expedited-forwarding	low
ef	“	high
af11	assured-forwarding	low
af12 (DSCP/EXP)	“	high
be	best-effort	low
be	“	high
nc1/cs6	network-control	low
nc2/cs7	“	high

See “Define Code-Point Aliases” on page 422 for the default bit definitions of DSCP, EXP, and IEEE code points.

To configure a rewrite-rules mapping and associate it with the appropriate forwarding class and code-point alias or bit set, include the rewrite-rules statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
rewrite-rules {
    (dscp | exp | inet-precedence) rewrite-name {
        import (rewrite-name | default);
        forwarding-class class-name {
            loss-priority (low | high) code-point alias | bits;
        }
    }
}
```

To assign the rewrite-rules configuration to the output logical interface, include the following configuration:

```
[edit class-of-service]
interfaces {
    interface-name {
        unit logical-unit-number {
            rewrite-rules {
                (dscp | exp | inet-precedence) rewrite-rule-name;
            }
        }
    }
}
```

You can include interface wildcards for interface-name and logical-unit-number.

You can include Layer 2 and Layer 3 rewrite information in the same configuration.

Configure CoS-Based Forwarding

CoS-based forwarding (CBF) enables you to control next-hop selection based on a packet's class of service and, in particular, the value of the IP packet's precedence bits.

For example, you might want to specify a particular interface or next hop to carry high-priority traffic while all best-effort traffic takes some other path. When a routing protocol discovers equal cost paths, it can pick a path at random or load-share across the paths either through hash selection or round robin. CBF allows path selection based on class.

You can apply CBF only to a defined set of routes. Therefore you must configure a policy statement as in the following example:

```
[edit]
policy-options {
    policy-statement my-cos-forwarding {
        from {
            route-filter filter-name;
        }
        then {
            cos-next-hop-map map-name;
        }
    }
}
```

This configuration specifies that routes matching the route filter will be subject to the CoS next-hop mapping specified by map-name. For more information about configuring policy statements, see the *JUNOS Internet Software Configuration Guide: Policy Framework*.

To specify a CoS next-hop map, include the forwarding-policy statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
forwarding-policy {
    next-hop-map map-name {
        forwarding-class class-name {
            next-hop [ next-hop-name ];
            lsp-next-hop [ lsp-regular-expression ]
        }
    }
}
```

The JUNOS software applies the CoS next-hop map to the set of next hops previously defined; the next hops themselves can be located across any outgoing interfaces on the router. For example, the following configuration associates a set of forwarding classes and next-hop identifiers:

```
[edit class-of-service forwarding-policy]
next-hop-map map1 {
        forwarding-class expedited-forwarding {
            next-hop next-hop1;
            next-hop next-hop2;
        }
        forwarding-class best-effort {
            next-hop next-hop3;
            lsp-next-hop lsp-next-hop4;
    }
}
```

In this example, next-hop*N* is either an IP address or an egress interface for some next hop, and lsp-next-hop4 is a regular expression corresponding to any next hop with that label. Q1 through Q*N* are a set of forwarding classes that map to the given next hop. That is, when a packet is switched with Q1 through Q*N*, it will be forwarded out the interface associated with the associated next hop.

This configuration has the following implications:

A single forwarding class can map to multiple standard next hops or LSP next hops. This implies that load sharing is done across standard next hops or LSP next hops servicing the same class value. To make this work properly, the software creates a list of the equal-cost next hops and forwards packets according to standard load-sharing rules for that forwarding class.

If a forwarding class configuration includes LSP next hops and standard next hops, the LSP next hops are preferred over the standard next hops. In the preceding example, if both next-hop3 and lsp-next-hop4 are valid next hops for a route to which map1 is applied, the forwarding table includes entry lsp-next-hop4 only.

If next-hop-map does not specify all possible forwarding classes, the default forwarding class is selected as the default. If the default forwarding class is not specified in the next-hop map, a default is designated randomly. The default forwarding class is the class associated with queue 0.

For LSP next hops, the JUNOS software uses UNIX regex(3)-style regular expressions. For example, if the following labels exist: lsp, lsp1, lsp2, lsp3, the statement lsp-next-hop lsp matches lsp, lsp1, lsp2, and lsp3. If you do not desire this behavior, you must use the anchor characters lsp-next-hop “^lsp$", which match lsp only.

The final step is to apply the route filter to routes exported to the forwarding engine. This is shown in the following example:

```
routing-options {
    forwarding-table {
        export my-cos-forwarding;
    }
}
```

This configuration instructs the routing process to insert routes to the forwarding engine matching my-cos-forwarding with the associated next-hop CBF rules.

The following algorithm is used when you apply a configuration to a route:

If the route is a single next-hop route, all traffic will go to that route; that is, no CBF will take effect.

For each next hop, associate the proper forwarding class. If a next hop appears in the route but not in the cos-next-hop map, it will not appear in the forwarding table entry.

The default forwarding class is used if all forwarding classes are not specified in the next-hop map. If the default is not specified, one is chosen randomly.

Override the Input Classification

For IPv4 or IPv6 packets, you can override the incoming classification, assigning them to the same forwarding class based on their input interface, input precedence bits, or destination address. You do so by defining a policy class when configuring CoS properties and referencing this class when configuring a routing policy.

When you override the classification of incoming packets, any mappings you configured for associated precedence bits or incoming interfaces to output transmission queues are ignored. Also, if the packet loss priority (PLP) bit was set in the packet by the incoming interlace, the PLP bit is cleared.

To override the input packet classification, do the following:

1. Define the policy class by including the class statement at the [edit class-of-service policy] hierarchy level:

   ```
   [edit class-of-service]
   forwarding-policy {
       class class-name {
           classification-override {
               forwarding-class class-name;
           }
       }
   }
   ```

   class-name is a name that identifies the class.

2. Associate the policy class with a routing policy by including it in a policy-statement statement at the [edit policy-options] hierarchy level. Specify the destination prefixes in the route-filter statement and the CoS policy class name in the then statement.

   ```
   [edit policy-options]
   policy-statement policy-name {
       term term-name {
           from {
               route-filter destination-prefix match-type <class class-name>;
           }
           then class class-name;
       }
   }
   ```

3. Apply the policy by including the export statement at the [edit routing-option] hierarchy level:

   ```
   [edit routing-options]
   forwarding-table {
       export policy-name;
   }
   ```

Example: Configure Class of Service

The following example includes classifiers, rewrite markers, and schedulers to configure a class of service policy.

1. Define a classifier that matches IP traffic arriving on the interface. The affected IP traffic has IP precedence bits with patterns matching those defined by aliases A or B. The loss priority of the matching packets is set to low, and the forwarding class is mapped to best effort (queue 0):

```
[edit]
class-of-service {
    classifiers {
        inet-precedence normal-traffic {
            forwarding-class best-effort {
                loss-priority low code-points [A B];
            }
        }
    }
}
```

   Following are the code-point alias and forwarding-class mappings referenced in the normal-traffic classifier:

```
[edit]
class-of-service {
    code-point-aliases {
        inet-precedence {
            A 000;
            B 001;
            ...
        }
    }
}

[edit]
class-of-service {
    forwarding-classes {
        queue 0 best-effort;
        queue 1 expedited-forwarding;
        queue 2 assured-forwarding;
        queue 3 network-control;
    }
}
```

2. Use rewrite markers to redefine the bit pattern of outgoing packets. Assign the new bit pattern based on specified forwarding classes, regardless of the loss priority of the packets:

```
[edit]
class-of-service {
    rewrite-rules {
        inet-precedence clear-prec {
            forwarding-class best-effort {
                loss-priority low code-point 000;
                loss-priority high code-point 000;
            }
            forwarding-class expedited-forwarding {
                loss-priority low code-point 100;
                loss-priority high code-point 100;
            }
        }
    }
}
```

3. Configure a scheduler map associating forwarding classes with schedulers and drop-profiles:

```
[edit]
class-of-service {
    scheduler-maps {
        one {
            forwarding-class expedited-forwarding scheduler special;
            forwarding-class best-effort scheduler normal;
        }
    }
}
```

Schedulers establish how to handle the traffic within the output queue for transmission onto the wire. Following is the scheduler referenced in scheduler map `one`:

```
[edit]
class-of-service {
    schedulers {
        special {
            transmit-rate percent 30;
            priority high;
        }
        normal {
            transmit-rate percent 70;
            priority low;
        }
    }
}
```

4. Apply the normal-traffic classifier to all SONET/SDH interfaces and all logical interfaces of SONET/SDH interfaces; apply the clear-prec rewrite marker to all Gigabit Ethernet interfaces and all logical interfaces of Gigabit Ethernet interfaces; and apply the one scheduler map to all interfaces:

```
[edit]
class-of-service {
    interfaces {
        so-* {
            unit * {
                classifiers {
                    inet-precedence normal-traffic;
                }
            }
        }
        ge-* {
            unit * {
                rewrite-rules {
                    inet-precedence clear-prec;
                }
            }
        }
        all {
            scheduler-map one;
        }
    }
}
```

Following is the complete configuration:

```
[edit class-of-service]
classifiers {
    inet-precedence normal-traffic {
        forwarding-class best-effort {
            loss-priority low code-points [A B];
        }
    }
}
code-point-aliases {
    inet-precedence {
        A 000;
        B 001;
        C 010;
        D 011;
        E 100;
        F 101;
        G 111;
        H 111;
    }
}
drop-profiles {
    high-priority {
        fill-level 20 drop-probability 100;
    }
    low-priority {
        fill-level 90 drop-probability 95;
    }
    big-queue {
        fill-level 100 drop-probability 100;
    }
}
```

```
forwarding-classes {
    queue 0 best-effort;
    queue 1 expedited-forwarding;
    queue 2 assured-forwarding;
    queue 3 network-control;
}
interfaces {
    so-* {
        unit * {
            classifiers {
                inet-precedence normal-traffic;
            }
        }
    }
    ge-* {
        unit * {
            rewrite-rules {
                inet-precedence clear-prec;
            }
        }
    }
    all {
        scheduler-map one;
    }
}
rewrite-rules {
    inet-precedence clear-prec {
        forwarding-class best-effort {
            loss-priority low code-point 000;
            loss-priority high code-point 000;
        }
        forwarding-class expedited-forwarding {
            loss-priority low code-point 100;
            loss-priority high code-point 100;
        }
    }
}
scheduler-maps {
    one {
        forwarding-class expedited-forwarding scheduler special;
        forwarding-class best-effort scheduler normal;
    }
}
schedulers {
    special {
        transmit-rate percent 30;
        priority high;
    }
    normal {
        transmit-rate percent 70;
        priority low;
    }
}
```